THE ACCELERATED BOUND-AND-SCAN ALGORITHM FOR INTEGER PROGRAMMING

by

Bruce H. Faaland

Frederick S. Hillier

TECHNICAL REPORT NO. 34

May 15, 1972

PREPARED UNDER CONTRACT

N00014-67-A-0112-0058 (NR-047-061)

FOR THE OFFICE OF NAVAL RESEARCH

Frederick S. Hillier, Project Director

This research was supported in part by
National ScienceFoundation Grant GJ-31521.

DEPARTMENT OF OPERATIONS RESEARCH

STANFORD UNIVERSITY

STANFORD, CALIFORNIA

The Accelerated Bound-and-Scan Algorithm for Integer Programming

by

Bruce H. Faaland and Frederick S. Hillier

1. Introduction

The second author [12] recently presented his Bound-and-Scan Algorithm for solving the pure integer linear programming (IP) problem,

$$\text{maximize} \quad c^T x \tag{1}$$

subject to

$$Ax \leq b \tag{2}$$

$$x \geq 0 \tag{3}$$

$$x \quad \text{integer}, \tag{4}$$

where A is an $m \times n$ matrix, b is an m-vector, and c is an n-vector. The encouraging computational experience with the algorithm suggests that the techniques proposed in [12] deserve further attention. Of particular interest is the extremely efficient procedure used to construct partial solutions for the "nonbasic" variables (nonbasic in the optimal linear programming solution), as described in Section 6. The remaining variables are called "basic" variables.

Given two problems of similar size and structure, the Bound-and-Scan Algorithm would generally be more efficient for the problem with fewer basic variables (and hence more nonbasic variables). This observation leads very naturally to the question: Is it possible to restructure the problem initially so that the restructured problem has fewer basic variables? The theory of equivalent

integer programs, introduced by G. Bradley [2] provides a partial answer to this question. This theory says that there exists an equivalent problem in which the number of basic variables is generally less than or otherwise equal to the number of basic variables in the original problem.

Solving an equivalent problem with fewer basic variables is one possibility for improving upon the Bound-and-Scan Algorithm. Another would be to extend to the basic variables the efficient bounding procedure for constructing partial solutions for the nonbasic variables. These two ideas provide the primary motivation for the basic enumeration scheme of the Accelerated Bound-and-Scan Algorithm to be described below.

Certain of the linear programming constraints (2,3) which hold with equality at $x^{(0)}$ are called binding constraints. To define these, recall that before the simplex algorithm is applied to the linear program (1,2,3), the inequality constraints (2) are first transformed to equality constraints

$$Ax + Is = b$$

by the addition of slack variables s. The binding constraints consist of those constraints among (2) whose slack variables are nonbasic in the optimal linear programming solution, and those among (3) which correspond to nonbasic variables at $x^{(0)}$. Thus, exactly n of the $(n+m)$ constraints (2,3) are binding constraints. Let these binding constraints be represented as

$$b_1 + A_1 x \geq 0. \qquad (5)$$

The other m constraints are called non-binding constraints and will be represented as

$$b_2 + A_2x \geq 0. \tag{6}$$

The integer program over the cone (the "cone problem") is the problem obtained from the integer programming problem by deleting the nonbinding constraints (6). The cone problem is therefore

$$\begin{aligned} &\text{maximize} && c^Tx \\ &\text{subject to} && b_1 + A_1x \geq 0 \\ & && x \quad \text{integer} . \end{aligned} \tag{7}$$

When the basic enumeration scheme is used to solve the integer program over the cone, the algorithm will be called the "cone algorithm." It will be shown that the cone algorithm is mathematically equivalent to the Bradley-Wahi enumeration algorithm [3] for the corresponding cone problem.

The feasible region for the cone problem is generally larger than the feasible region for the integer programming problem. Any algorithm designed to solve the more restricted problem must consider the additional constraints present in the integer program. Three techniques are developed in Section 8 for dealing with these constraints. The first of these techniques is the standard device of considering only values of variables between fixed upper and lower bounds. The bounds are calculated once at the outset and are based on all the constraints of the original problem rather than on just the cone constraints. The second method of dealing with the additional constraints of the IP problem is to use a scanning procedure whenever an infeasible solution within the cone is generated. The scanning procedure allows the algorithm to "skip over" possibly many

other infeasible solutions in the basic enumeration scheme. The third technique is a means of recognizing and eliminating redundant constraints. Constraints which must be considered at the start of the algorithm may become superfluous as improved solutions to the IP problem are found.

When the cone algorithm is supplemented by these three techniques, the resulting algorithm will be called the "skeleton version" of the Accelerated Bound-and-Scan Algorithm. Additional options for the algorithm also are available. In particular, conditional bounds on variables might be calculated by techniques suggested in [12] or in [5, Ch. VII]. Surrogate constraints [7] might be very useful in determining the feasibility of solutions generated within the cone.

As in [12], the two main assumptions used by the Accelerated Bound-and-Scan Algorithm are first, that the optimal noninteger solution $x^{(0)}$ to (1,2,3) is unique, and second, that a good feasible integer solution $x^{(F)}$ to (1,2,3,4) has already been identified. Efficient heuristic procedures for obtaining $x^{(F)}$ are available [13] when (2,3) possess interior points. Otherwise, the system of Diophantine equations must first be solved to eliminate the equality constraints An additional assumption made here is that all elements of A,b and c are integer-valued.

In Section 2 the theory of equivalent integer programs is reviewed. The fundamental congruence system used by the algorithm is derived in Section 3. Section 4 presents the basic enumeration scheme, and Section 5 outlines the cone algorithm In Section 6 the special computational property of the nonbasic variables is displayed, and in Section 7 the cone algorithm is compared with the Bradley-Wahi cone algorithm. Section 8 presents the skeleton version of the Accelerated Bound-and-Scan Algorithm, and computational experience is given in Section 9, followed by conclusions in Section 10.

2. Equivalent Integer Programs and the Cone Problem

Bradley [2] has shown that the integer programming problem is equivalent to infinitely many other integer programming problems of the form

$$\begin{aligned} \text{maximize} \quad & c^T e + (c^T K)y \qquad (8) \\ \text{subject to} \quad & (AK)y \leq b - Ae \\ & -K y \leq e \\ & y \quad \text{integer}, \end{aligned}$$

where K is an $n \times n$ unimodular matrix (a square integer matrix whose determinant equals plus or minus one), and e is an integer n-vector. Problem (8) may be solved instead of problem (1,2,3,4). The correspondence between the solutions x and y is given by

$$x = Ky + e \qquad (9)$$

and

$$y = K^{-1}(x-e). \qquad (10)$$

The following results are fundamental in the selection of the equivalent problem (8) to be solved.

Theorem 1 (Hermite [2]): Given an $n \times n$ integer matrix C of full rank, there exists an $n \times n$ unimodular matrix K such that CK is lower triangular with positive diagonal elements, and each off-diagonal element is nonpositive and strictly less in absolute value than the diagonal element in its row.

CK is called the Hermite normal form of C.

Theorem 2 (Bradley [2]): The integer programming problem (1,2,3,4) is equivalent to an integer programming problem such that in the new problem the $n \times n$ coefficient matrix of the binding constraints is in Hermite normal form and each element of the constant column of the binding constraints is nonnegative and strictly less than the corresponding diagonal element of the coefficient matrix.

The equivalent problem mentioned in Theorem 2 is of the form,

$$\begin{aligned} &\text{maximize} && c^T e + (c^T K)y && (11)\\ &\text{subject to} && (b_1 + A_1 e) + A_1 K y \geq 0 \\ & && (b_2 + A_2 e) + A_2 K y \geq 0 \\ & && y \quad \text{integer,} \end{aligned}$$

where $A_1 K$ is the Hermite normal form of A_1.

In the new variables y the cone problem is

$$\begin{aligned} &\text{maximize} && c^T e + (c^T K)y && (12)\\ &\text{subject to} && (b_1 + A_1 e) + (A_1 K)y \geq 0 \\ & && y \quad \text{integer.} \end{aligned}$$

There are reasons for choosing K so that $A_1 K$ is the Hermite normal form of A_1. The matrix A_1 is singled out because in general the binding constraints are the constraints most likely to be violated by an integer solution in the vicinity of the optimal linear programming solution. In the Accelerated Bound-and-Scan Algorithm, the binding constraints serve as a filter for examining possible solutions. It may therefore be desirable to select the equivalent problem so that the binding constraints have as "simple" a form as possible. The lower triangular property of

the Hermite normal form makes it feasible to apply either the Fourier-Motzkin elmination proceaure suggested by Bradley and Wahi [3] or the Accelerated Bound-and-Scan Algorithm to be described below. Another appealing property of the Hermite normal form is that typically many of the diagonal elements of the integer matrix A_1K will be unit elements. Each unit element along the diagonal of A_1K corresponds to a simple nonnegativity constraint $y_i \geq 0$, and hence to a nonbasic variable in the optimal linear programming solution. If the rows of A_1 are initially ordered so that the binding nonnegativity constraints of the original problem precede the binding functional constraints, the equivalent cone problem (12) will have at least as many nonnegativity constraints (and hence nonbasic variables) as the original cone problem, and usually many more. This ordering of the rows of A_1 will be assumed (as the most convenient one) for the algorithm.

The phenomenon of many unit elements along the diagonal of A_1K is consistent with a well know interpretation of the procedure for obtaining the Hermite normal form of a matrix. The procedure may be viewed as successive applications of the Euclidean algorithm for finding the greatest common divisor (gcd) of two integers. For example, the first diagonal element of the Hermite normal form A_1K is the gcd of the n integers in the first row of A_1. In general, the j^{th} diagonal element of A_1K is the gcd of $(n-j+1)$ integers. The following simple result may make plausible the occurrence of unit elements along the diagonal of the Hermite normal form A_1K if the elements of A_1 are generated randomly. The notation $(k_1,\cdots,k_t)$ denotes the gcd of $k_1,\cdots,k_t$, and $[z]$ denotes the greatest integer less than or equal to z.

Theorem 3 Let $k_1,\ldots,k_t$ be independent, identically distributed random variables with the probability mass function

$$P(k_j = s) = \frac{1}{2m} \qquad s = \pm 1, \pm 2, \ldots, \pm m,$$

for all $j = 1,\ldots,t$. Then

$$P((k_1,\ldots,k_t) = 1) \geq 1 - \sum_{i=2}^{m} ([m/i]/m)^t. \tag{13}$$

Proof

$$P((k_1,\ldots,k_t) = 1) = 1 - \sum_{i=2}^{m} P((k_1,\ldots,k_t) = i)$$

$$\geq 1 - \sum_{i=2}^{m} P(i \text{ divides every } k_j,\ j = 1,\ldots,t)$$

$$= 1 - \sum_{i=2}^{m} ([m/i]/m)^t$$

Consider as an example the case where $m = 6$ and $t = 5$. Then by (13) the probability that five randomly generated integers have gcd equal to one is at least .96.

3. The Fundamental Congruence System

If the objective function value z is treated as a parameter, the equivalent cone problem (12) can be replaced by the inequality system in integer variables $y_1,\ldots,y_n$:

$$\begin{aligned} g_1y_1 + g_2y_2 + \ldots + g_ny_n &\geq z - c^Te \\ h_{11}y_1 &\geq \beta_1 \\ h_{21}y_1 + h_{22}y_2 &\geq \beta_2 \\ &\vdots \\ h_{n1}y_1 + h_{n2}y_2 + \ldots + h_{nn}y_n &\geq \beta_n, \end{aligned} \tag{14}$$

where

$$\begin{aligned} h_{ij} &= (A_1K)_{ij} && i,j = 1,\ldots,n, \\ \beta_i &= (b_1+A_1e)_i && i = 1,\ldots,n, \\ g_j &= (c^TK)_j && j = 1,\ldots,n. \end{aligned}$$

The Accelerated Bound-and-Scan Algorithm is based on system (14). For a fixed value of z corresponding to a known feasible solution, the $(n+1)$ constraints of (14) define an n-simplex S (in Euclidean n-space) in which the optimal solution to the integer programming problem must lie. The n-simplex S is a convex set, and therefore every point in S can be expressed as a convex combination of the vertices $y^{(i)}$ $(i = 0,\ldots,n)$ of S. That is, for every y in S there exist

unique extreme point weights ρ_i $(i = 0,1,\cdots,n)$ such that

$$y = \sum_{i=0}^{n} \rho_i y^{(i)} \tag{15}$$

$$\sum_{i=0}^{n} \rho_i = 1$$

$$\rho_i \geq 0 \qquad i = 0,1,\cdots,n.$$

An equivalent representation would be

$$y = y^{(0)} + \sum_{i=1}^{n} \rho_i (y^{(i)} - y^{(0)}) \tag{16}$$

$$\sum_{i=1}^{n} \rho_i \leq 1$$

$$\rho_i \geq 0 \qquad i = 1,\cdots,n.$$

The integer requirement on y is satisfied by considering only extreme point weights $\rho_1,\cdots,\rho_n$ which satisfy the congruence system

$$y^{(0)} + \sum_{i=1}^{n} \rho_i (y^{(i)} - y^{(0)}) \equiv 0 \pmod 1 \tag{17}$$

$$\sum_{i=1}^{n} \rho_i \leq 1$$

$$\rho_i \geq 0 \qquad i = 1,\cdots,n.$$

Lemma 1 The coefficient matrix of (17) is lower triangular with strictly positive diagonal entries. That is, (17) is of the form:

$$\begin{pmatrix} d_1 \\ \vdots \\ d_n \end{pmatrix} + \begin{pmatrix} B_{11} & & \\ \vdots & \ddots & \\ B_{n1} & \cdots & B_{nn} \end{pmatrix} \begin{pmatrix} \rho_1 \\ \vdots \\ \rho_n \end{pmatrix} \equiv 0 \pmod 1 \tag{18}$$

$$\sum_{i=1}^{n} \rho_i \leq 1$$

$$\rho_i \geq 0 \qquad i = 1,\ldots,n,$$

where $B_{ii} > 0$ for $i = 1,\ldots,n$.

Proof The linear programming solution $y^{(0)}$ is found by dropping the objective function constraint

$$g_1y_1 + g_2y_2 + \cdots + g_ny_n \geq z - c^Te$$

from (14) and solving the simultaneous equation system

$$\begin{aligned} h_{11}y_1 &= \beta_1 \\ h_{21}y_1 + h_{22}y_2 &= \beta_2 \\ &\vdots \\ h_{n1}y_1 + h_{n2}y_2 + \cdots + h_{nn}y_n &= \beta_n \end{aligned} \tag{19}$$

Since (19) is a lower triangular equation system, the first j components $y_1^{(0)},\cdots,y_j^{(0)}$ of $y^{(0)}$ are uniquely determined by the first j equations of (19).

The extreme point $y^{(i)}$, $i = 1,\cdots,n$, is the unique solution to the equation system

$$\begin{aligned} h_{11}y_1 &= \beta_1 \\ &\vdots \\ h_{i-1,1}y_1 + \cdots + h_{i-1,i-1}y_{i-1} &= \beta_{i-1} \\ g_1y_1 + \cdots + g_ny_n &= z-c^Te \\ h_{i+1,1}y_1 + \cdots + h_{i+1,i+1}y_{i+1} &= \beta_{i+1} \\ &\vdots \\ h_{n1}y_1 + \cdots + h_{nn}y_n &= \beta_n, \end{aligned} \tag{20}$$

obtained from (19) by replacing the i^{th} equation of (19) by the equation

$$g_1y_1 + \cdots + g_ny_n = z - c^Te .$$

Since the first (i-1) equations of (20) form a lower triangular system identical with the first (i-1) equations of (19), $(y_j^{(i)}-y_j^{(0)}) = 0$ for all $j = 1,\cdots,i-1$, so the constraint matrix of (17) is lower triangular.

To show that $B_{ii}>0$, it is sufficient to show that ρ_i is an increasing function of y_i. From equation (44) in [12], it follows that

$$\rho_i = (\sum_{j=1}^{i-1} h_{ij}y_j-\beta_i)/Q_i + (h_{ii}/Q_i)y_i,$$

where

$$Q_i \equiv \sum_{j=1}^{i} h_{ij}y_j^{(i)} - \beta_i.$$

By the construction of (14), $h_{ii}>0$, and by Lemma 3 in [12], $Q_i>0$. Therefore $B_{ii}>0$, and the proof of Lemma 1 is complete.

It should be pointed out that the congruence system (18) is not unique. Each of the n! orderings of the rows of A_1 yields a (possibly) different system (14), which in turn determines (18). The ordering of the rows of A_1 may play an important role in the computational efficiency of the algorithm.

4. The Basic Enumeration Scheme

The basic enumeration scheme for the Accelerated Bound-and-Scan Algorithm is a systematic way of enumerating implicitly all solutions to congruence system (18), and hence all lattice points within the n-simplex. Such an implicit enumeration scheme may be used to solve the cone problem. The "cone algorithm" proceeds by considering sequences of partial solutions to system (18).

An "eligible partial solution" through variable ρ_j is a specification $(\rho_1^*,\cdots,\rho_j^*)$ of the values of $\rho_1,\cdots,\rho_j$ which satisfies

$$\begin{pmatrix} d_1 \\ \vdots \\ d_j \end{pmatrix} + \begin{pmatrix} B_{11} & & \\ \vdots & \ddots & \\ B_{j1} & \cdots & B_{jj} \end{pmatrix} \begin{pmatrix} \rho_1 \\ \vdots \\ \rho_j \end{pmatrix} \equiv 0 \pmod 1 \qquad (21)$$

$$\sum_{i=1}^{j} \rho_i \leq 1$$

$$0 \leq \rho_i \leq \bar{\rho}_i \qquad i = 1,\cdots, j.$$

The term $\bar{\rho}_i$ is an upper bound on the maximum weight which may be placed on the i^{th} extreme point and still yield a solution which satisfies both the binding and nonbinding constraints. In general $\bar{\rho}_i \leq 1$, and $\bar{\rho}_i < 1$ if the i^{th} extreme point violates a nonbinding constraint. For the cone problem $\bar{\rho}_i = 1$ for all $i = 1,\cdots,n$ since the nonbinding constraints are not present in the problem. The determination of the $\bar{\rho}_i$ is discussed in Section 8. Eligible partial solutions through ρ_n are called "completions," for they completely specify a lattice point within S.

Suppose an eligible partial solution $(\rho_1^*,\cdots,\rho_{k-1}^*)$ through ρ_{k-1} has been specified. The algorithm must determine by some procedure whether there exists a value for ρ_k such that $(\rho_1^*,\cdots,\rho_{k-1}^*,\rho_k)$ is an eligible partial solution through ρ_k. This procedure is called a "forward step" in the algorithm. If f denotes the fractional part of $d_k+\sum_{i=1}^{k-1} B_{ki}\rho_i^*$, i.e.,

$$f = d_k + \sum_{i=1}^{k-1} B_{ki}\rho_i^* - \left[d_k+\sum_{i=1}^{k-1} B_{ki}\rho_i^*\right], \qquad (22)$$

ρ_k must be chosen to satisfy

$$f + B_{kk}\rho_k \equiv 0 \pmod 1 \tag{23}$$

$$\sum_{i=1}^{k-1}\rho_i^* + \rho_k \leq 1 \tag{24}$$

$$0 \leq \rho_k \leq \bar{\rho}_k. \tag{25}$$

Consider first the integer restriction (23), and let t denote the smallest value ρ_k may take on and still satisfy (25). If $f = 0$, then $\rho_k = 0$ satisfies (23); clearly $t = 0$ in this case. Recall from Lemma 1 that $B_{kk}>0$. If $0<f<1$, then t must be chosen so that $f+B_{kk}t = 1$. Therefore

$$t = \begin{cases} (1-f)/B_{kk} & \text{if } 0 < f < 1 \\ 0 & \text{if } f = 0. \end{cases} \tag{26}$$

From (23) the only possible values ρ_k can take on , given the eligible partial solution $(\rho_1^*,\cdots,\rho_{k-1}^*)$, are

$$\rho_k = t, t + 1/B_{kk},\cdots,t + r/B_{kk}\ , \tag{27}$$

where r is the largest integer such that

$$t + r/B_{kk} \leq \bar{\rho}_k\ .$$

The algorithm sets ρ_k at t and checks conditions (24) and (25). If either (24) or (25) is violated, it is obvious that no larger value of ρ_k would satisfy these conditions, and hence there can be no completion for the partial solution $(\rho_1^*,\cdots,\rho_{k-1}^*)$. The algorithm would then backtrack to search for a new eligible partial solution

through ρ_{k-1} by resetting ρ_{k-1} at $\rho^*_{k-1}+1/B_{k-1,k-1}$ and checking simply whether

$$\sum_{i=1}^{k-1} \rho_i^* + 1/B_{k-1,k-1} \leq 1 \qquad (28)$$

$$\rho^*_{k-1} + 1/B_{k-1,k-1} \leq \bar{\rho}_{k-1} .$$

Notice that if the terms $1/B_{ii}$ $(i = 1,\ldots,n)$ are stored, the partial sum of extreme point weights can be updated by only an addition and/or a subtraction each time a forward or a backtrack step is made.

Suppose $\rho_k = t$ satisfies conditions (24) and (25), so that an eligible partial solution through variable ρ_k has been found. The algorithm then attempts to construct an eligible partial solution through ρ_{k+1} and continues the forward steps until either a completion within the n-simplex has been found (an improved solution), or an eligible partial solution through the next variable cannot be found. In the latter case the algorithm backtracks until the next eligible partial solution is found before starting the forward steps again. When the algorithm backtracks to ρ_1, and no new eligible partial solution through ρ_1 exists, the algorithm terminates.

The integer programming problem may be posed (see Theorem 2 in [12]) as the problem of finding the feasible lattice point which places the greatest weight on the extreme point $x^{(0)}$. If a completion has been generated, it must be checked for feasibility in the general integer programming problem against the nonbinding constraints. This process is discussed in detail in Section 8. Since there are no nonbinding constraints in the cone problem, every completion is a feasible completion for the cone problem. In either problem, once a feasible completion has been found, the algorithm backtracks to ρ_{n-1}. Every subsequent solution generated by the algorithm is required to have a larger objective function value than

that of this improved solution. A necessary condition for any partial solution $(\rho_1^*, \ldots, \rho_j^*)$ to be an eligible partial solution is that

$$\sum_{i=1}^{j} \rho_i^* \leq \left(\sum_{i=1}^{n} \tilde{\rho}_i\right), \tag{29}$$

where $(1-\sum_{i=1}^{n} \hat{\rho}_i)$ is the value of ρ_0 for the best feasible solution $(\hat{\rho}_1, \ldots, \hat{\rho}_n)$ known at the current stage of the algorithm. Condition (29) replaces the condition

$$\sum_{i=1}^{j} \rho_i \leq 1$$

in the definition of an eligible partial solution once a feasible completion $(\tilde{\rho}_1, \ldots, \tilde{\rho}_n)$ has been found.

5 The Cone Algorithm

The cone algorithm is the basic enumeration scheme applied to the integer program over the cone. Since $\bar{\rho}_j = 1$ $(j = 1, \ldots, n)$ in the cone problem, the condition $\rho_j \leq \bar{\rho}_j$ is superfluous and is not used in the step-by-step description given below.

Outline of Cone Algorithm

Step 1. Set $B_j^* = 1/B_{jj}$ for $j = 1, \ldots, n$. Set Best = 1, Sum = 0, $k = 1$, and go to Step 3.

Step 2. Reset $k = k+1$. If $k>n$, go to Step 8.

Step 3. Let f be the fractional part of $d_k + \sum_{i=1}^{k-1} B_{ki}\rho_i$.

Step 4. If $f = 0$, set $\rho_k = 0$ and go to Step 2. Otherwise set $\rho_k = (1-f)B_k^*$.

Step 5. Set Sum1 = Sum + ρ_k.

Step 6. If Sum1 $\geq$ Best, go to Step 7. Otherwise reset Sum = Sum1 and go to Step 2.

Step 7. Reset $k = k-1$. If $k = 0$, stop. Otherwise reset $\text{Sum} = \text{Sum}-\rho_k$ and $\rho_k = \rho_k+B_k^*$. Set $\text{Sum1} = \text{Sum}+\rho_k$, and go to Step 6.

Step 8. An improved completion has been found. Store the solution $(\rho_1,\cdots,\rho_n)$. Reset $k = n$, $\text{Sum} = \text{Sum}-\rho_n$, and $\text{Best} = \sum_{j=1}^{n}\rho_j$. Go to Step 7.

6. A Special Property of the Nonbasic Variables

Assume the y_k $(k = 1,\ldots,n)$ are ordered so that the first n_1 variables correspond to nonnegativity constraints in the equivalent cone problem (12). Each of the y_k $(k = 1,\ldots,n_1)$ would be nonbasic in the linear programming solution, and as noted in [12], each y_k $(k = 1,\ldots,n_1)$ has the value zero at all extreme points $y^{(i)}$ of the n-simplex except one which is designated $y^{(k)}$. That is, $y_k^{(i)} = 0$ for all $i = 0,1,\ldots,n$, $i \neq k$, and $y_k^{(k)}>0$. An immediate consequence of this property is that the coefficients of the first n_1 rows of congruence system (18) have the following special form. For $k = 1,\ldots,n_1$,

$$d_k = 0 \tag{30}$$

$$B_{kk} = y_k^{(k)}$$

$$B_{kj} = 0 \qquad j = 1,\ldots,k-1$$

Therefore

$$\rho_k = y_k/B_{kk} \qquad k = 1,\ldots,n_1. \tag{31}$$

Compare this expression with (27). If $1\leq k\leq n_1$, the fractional part f of (22) is zero, so by (26) $t = 0$, and (27) reduces to (31).

From a computational point of view, the essential difference between those extreme point weights ρ_k which are associated with nonbasic variables and those which are not is that for the former the fractional part is known to be zero, whereas for the latter f must be calculated. The calculation of f at each forward step is by far the most expensive part of the cone algorithm, and should be avoided whenever possible. Given the special property (31), Step 3 of the Cone Algorithm Outline should therefore be revised to read:

Step 3'. If $k \leq n_1$, set $\rho_k = 0$ and go to Step 2. Otherwise let f be the fractional part of $d_k + \sum_{i=1}^{k-1} B_{ki}\rho_i$.

This technique is equivalent to the procedure for constructing partial solutions for the nonbasic variables in the original Bound-and-Scan Algorithm. Thus the goal of extending the efficient bounding procedure for the nonbasic variables to the basic variables as well as been attained - with the one essential computational difference described above.

7. Comparison with the Bradley-Wahi Cone Algorithm

The implicit enumeration algorithm for the cone problem devised by Bradley and Wahi [3] is also based on (14). To this system Bradley and Wahi apply the Fourier-Motzkin elimination method, elminating the variables in the order $y_n, y_{n-1}, \ldots, y_1$. As a result of this elimination procedure, conditional upper and lower bounds on each variable (with respect to the cone) may be calculated relatively easily. The continuous upper bound on y_i, given that the first (i-1) variables are fixed at $y_1^*, \ldots, y_{i-1}^*$, is a linear function $U_i(\cdot)$ of $z, y_1^*, \ldots, y^*_{i-1}$. The conditional lower bound is a linear function $L_i(\cdot)$ of only $y_1^*, \ldots, y_{i-1}^*$.

Suppose that $y_1,\ldots,y_{n_1}$ correspond to nonnegativity constraints in the equivalent cone problem. It is possible to show for $i \leq n_1$ that $U_i(\cdot)$ can be calculated from $U_{i-1}(\cdot)$, using only a fixed number of arithmetic operations independent of i, and that $L_i(\cdot) \equiv 0$. For $i = n_1+1,\ldots,n$, there appears to be no way to reduce the number of multiplications and additions, $2(i-1)$, required to compute $U_i(\cdot)$ and $L_i(\cdot)$.

It is now possible to show that the cone algorithm and the Bradley-Wahi algorithm are mathematically equivalent in the sense that they will generate the same sequence of eligible partial solutions for a given cone problem (14).

There is a one-to-one correspondence between partial solutions in variables $\rho_1,\ldots,\rho_k$ and partial solutions in variables $y_1,\ldots,y_k$. An eligible partial solution in either variable system represents a lattice point within the k-simplex formed by the projection of the n-simplex onto the space of the first k variables $y_1,\ldots,y_k$.

Assuming the same ordering of the y_k variables in each algorithm, it will be shown that eligible partial solutions are constructed in the same order in both algorithms. Consider first the forward step. Assume that the algorithms have each constructed an eligible partial solution through variable k which corresponds to the same lattice point in the k-simplex. Each algorithm attempts to construct an eligible partial solution through variable $(k+1)$ by setting variable $(k+1)$ at its calculated lower bound with respect to the projected $(k+1)$ simplex. By the proof of Lemma 1, ρ_{k+1} is an increasing function of y_{k+1}. Therefore, the lower bound value for ρ_{k+1} must correspond to the lower bound value for y_{k+1}, and the partial solutions generated by the algorithm represent the same lattice point. Next, consider the backtrack step.

When the Bradley-Wahi algorithm backtracks to y_k, it increases y_k to (y_k+1), which is equivalent to increasing ρ_k to $(\rho_k+B_k^*)$ as in Step 7 of the cone algorithm. Thus, the cone algorithm and the Bradley-Wahi algorithm are mathematically equivalent.

The computational effort required by each algorithm will be compared next. Since partial solutions are constructed in the same order, it is sufficient to compare the work required on forward and backtrack steps. The two algorithms seem to require the same computational effort on forward and backtrack steps involving ρ_k for $k \leq n_1$. For $k > n_1$ the Bradley-Wahi algorithm has the advantage in the backtrack step because it needs only to reset y_k at y_k+1 and check whether the upper bound U_k has been violated. The cone algorithm, on the other hand, requires another two additions in Step 7 to adjust Sum besides the addition required to reset ρ_k at $\rho_k+B_k^*$.

However, the cone algorithm is superior on the more expensive forward steps. The Bradley-Wahi algorithm requires $2(k-1)$ multiplications and additions on a forward step to calculate L_k and U_k. L_k is calculated to determine the value at which y_k should be set, and U_k is needed to decide whether this value of y_k results in a eligible partial solution. The cone algorithm uses k multiplications and additions to find the lower bound for ρ_k and verify whether it results in a eligible partial solution. Note that the total number of forward steps must equal the total number of backtrack steps The two algorithms therefore seem to be roughly comparable in computational effort needed to solve the cone problem.

It should be emphasized at this point that the cone problem, because of its fewer constraints and simpler form, is easier to solve than the original integer programming problem (1,2,3,4). Furthermore,

the Bradley-Wahi algorithm deals only with the cone problem. The Accelerated Bound-and-Scan Algorithm has several features which enable it to deal effectively with the nonbinding constraints, and therefore with the original integer program.

One of these features, the scanning procedure, uses to advantage the unit coefficient on the variable ρ_i in the relationship

$$\sum_{i=1}^{n} \rho_i \leq 1 \tag{32}$$

among $\rho_1,\ldots,\rho_n$. The upper bounds and estimates of computational effort given in a companion paper [4] are also derived using (32). Although it might be possible to perform the same type of analysis in variables $y_1,\ldots,y_n$, the work is simplified by the unit coefficient property of (32). Certainly the scanning procedure is more efficient because of this property.

8. The Skeleton Version of the Accelerated Bound-and-Scan Algorithm

Since the n-simplex S is determined by only the binding constraints and the additional objective function constraint in (14), S may include points which violate a nonbinding constraint, and are therefore not feasible for the original integer programming problem (1,2,3,4). To exclude from consideration some (but not necessarily all) of these infeasible points, upper bounds $\bar{\rho}_i$ on the extreme point weights ρ_i may be found by linear programming or by ad hoc methods suggested in [12].

The addition of upper bounds on the ρ_i is one of three techniques in the "skeleton version" of the Accelerated Bound-and-Scan Algorithm for dealing with nonbinding constraints. The second technique is a scanning procedure. Suppose a completion $(\rho_1^*,\ldots,\rho_n^*)$ which satisfies all the conditions in (21) has been generated by the algorithm If $(\rho_1^*,\ldots,\rho_n^*)$ satisfies the nonbinding constraints, a new improved solution has been

found. If not, the algorithm could proceed by generating the next completion according to the enumeration scheme. However, if the current completion violates a certain nonbinding constraint, there is a good chance this constraint will also be violated by the next several completions that would be generated by the enumeration scheme. The purpose of the scanning procedure is to determine the next eligible partial solution which can possibly lead to a feasible completion, so that the enumeration scheme can be restarted from that point.

The m nonbinding constraints (9) can be represented in the ρ_i variables through changes in variables given by (6) and (16) as

$$\sum_{i=1}^{n} -A_2K(y^{(i)}-y^{(0)})\rho_i \leq -(b_2+A_2e+A_2Ky^{(0)}). \tag{33}$$

Let

$$\sum_{i=1}^{n} T_i\rho_i \leq T \tag{34}$$

be any one of the m nonbinding constraints, and let $(\rho_1^*,\ldots,\rho_n^*)$ be a completion which has been found to violate (34). Then the scanning procedure may be summarized as follows. (In Step 7 of the outline, the notation $\langle t \rangle$ designates the smallest integer greater than or equal to t.)

Outline of the Scanning Procedure

Step 1. Set $k = n$

Step 2. Set $\gamma = \arg \min_{i=k,\ldots,n} T_i$. If $T_\gamma<0$, go to Step 5.

Step 3. If $\sum_{i=1}^{k-1} T_i\rho_i^* \leq T$, go to Step 8.

Step 4. Reset $k = k-1$. If $k = 0$, algorithm terminates. Otherwise, go to Step 2.

Step 5. If $\sum_{i=1}^{k-1} T_i\rho_i^* + (1-\sum_{i=1}^{k-1} \rho_i^*) T_\gamma > T$, go to Step 4.

Step 6. If $T_k \neq T_\gamma$, go to Step 8.

Step 7. Set $u^* = (T-\sum_{i=1}^{k-1} T_i\rho_i^*)/T_k$.

Set $r = \langle(u^*-\rho_k^*)B_{kk}\rangle$ and reset $\rho_k^* = \rho_k^*+r/B_{kk}$.

Exit scanning procedure.

Step 8. Reset $k = k-1$. If $k = 0$, algorithm terminates. Otherwise, reset $\rho_k^* = \rho_k^* + B_k^*$.

Exit scanning procedure.

The heart of the scanning procedure lies in the tests performed in Step 3 and Step 5. At each of these steps it has already been determined that the algorithm must backtrack at least to variable ρ_k in order to find the next eligible partial solution which can possibly lead to a completion satisfying (34). If $\rho_1,\ldots,\rho_{k-1}$ are fixed in a partial solution $(\rho_1^*,\ldots,\rho_{k-1}^*)$, up to $(1-\sum_{i=1}^{k-1}\rho_i^*)$ may be allocated to the extreme point weights $\rho_k,\ldots,\rho_n$ in any possible completion. Suppose this is allocated in the most favorable way toward satisfying (34). Zero weight would be assigned to $\rho_k,\ldots,\rho_n$ (as in Step 3) if $T_\gamma \geq 0$. And if $T_\gamma < 0$, all the weight $(1-\sum_{i=1}^{k-1}\rho_i^*)$ would be assigned to T_γ (as in Step 5). This assignment provides, of course, a test which is only a necessary condition for the existence of a feasible completion of a partial solution. The scanning procedure constructs the next partial solution which satisfies this test.

The importance of using a scanning procedure in conjunction with the nonbinding constraints was pointed out in [12]. The procedure discussed in this section is based on the scanning procedure used in the original Bound-and-Scan Algorithm.

The "standard procedure" suggested here is to stop at the first violated nonbinding constraint, construct the next partial solution which can possibly lead to a completion satisfying the violated constraint, and

then restart the enumeration scheme at this new partial solution. A possible variant to this approach would be to use arguments similar to those in Step 3 and Step 5 to determine whether this new partial solution can possibly lead to a completion which satisfies the nonbinding constraints that have not yet been checked.

The third technique for taking the nonbinding constraints into account is the elimination of redundant nonbinding constraints at various points in the algorithm. A nonbinding constraint (34) which is satisfied by every subsequent completion within the n-simplex S is called a redundant constraint. It may be dropped from the problem, thus eliminating the arithmetic operations required to check it for feasibility.

Redundant constraints are easily recognizable in the formulation of the II problem which uses variables $\rho_1,\ldots,\rho_n$. Let $(\bar{\rho}_1,\ldots,\bar{\rho}_n)$ be the best feasible solution known at the current stage of the algorithm, and let $\bar{\sigma} \equiv \sum_{i=1}^{n}\bar{\rho}_i$. Consider any nonbinding constraint (34), and let $T^* \equiv \max_{i=1,\ldots,n} T_i$. If $T^* \leq 0$, constraint (34) is redundant, and if $T^* > 0$, a sufficient condition for constraint (34) to be redundant is that

$$\bar{\sigma} \leq T/\max_{i=1,\ldots,n} T_i \qquad (35)$$

After calculating the right-hand side once at the outset, condition (35) would be checked every time an improved feasible solution has been generated by the algorithm.

An outline of the skeleton version of the Accelerated Bound-and-Scan Algorithm follows. It incorporates the cone algorithm, the special procedure for nonbasic variables, upper bounds on variables, the scanning procedure, and the elimination of redundant constraints. The scanning procedure outline is used as a subroutine in Step 9.

Outline of Skeleton Algorithm

Step 1. Set $B_j^* = 1/B_{jj}$ for $j = 1,\ldots,n$. Set Best = 1, Sum = 0, $k = 1$, and go to Step 3.

Step 2. Reset $k = k+1$. If $k>n$, go to Step 8.

Step 3. If $k \leq n_1$ set $\rho_k = 0$ and go to Step 2. Otherwise let f be the fractional part of $d_k + \sum_{i=1}^{k-1} B_{ki}\rho_i$.

Step 4. If $f = 0$, set $\rho_k = 0$ and go to Step 2. Otherwise set $\rho_k = (1-f)B_k^*$.

Step 5. Set Sum1 = Sum+ρ_k

Step 6. If Sum1 $\geq$ Best or if $\rho_k > \bar{\rho}_k$, go to Step 7. Otherwise reset Sum = Sum1 and go to Step 2.

Step 7. Reset $k = k-1$. If $k = 0$, algorithm terminates. Otherwise reset Sum = Sum-ρ_k and $\rho_k = \rho_k + B_k^*$. Set Sum1 = Sum+ρ_k, and go to Step 6.

Step 8. Check the nonbinding constraints until one is found which is violated by $(\rho_1,\ldots,\rho_n)$. If all nonbinding constraints are satisfied, go to Step 9. Otherwise go to Step 10.

Step 9. An improved solution has been found. Store the solution $(\rho_1,\ldots,\rho_n)$. Reset $k = n$, Sum = Sum-ρ_n, and Best = $\sum_{j=1}^{n}\rho_j$. Eliminate redundant nonbinding constraints and go to Step 7.

Step 10. Represent the violated nonbinding constraint as

$$\sum_{i=1}^{n} T_i\rho_i \leq T.$$

Set $(\rho_1^*,\ldots,\rho_n^*) = (\rho_1,\ldots,\rho_n)$. Enter the scanning procedure. The scanning procedure will either indicate that the algorithm terminates, or it will produce a new partial solution $(\rho_1^*,\ldots,\rho_k^*)$. Set $(\rho_1,\ldots,\rho_k) = (\rho_1^*,\ldots,\rho_k^*)$ and Sum = $\sum_{i=1}^{k-1}\rho_i$. Go to Step 5.

The algorithm outlined above is called the skeleton algorithm because it includes only the three simple techniques for handling the nonbinding constraints: bounds on variables, the scanning procedure, and the elimination of redundant constraints. It is suggested that these three techniques always be used. Additional options which may be used include conditional bounds on variables (as discussed in [12] or in [5, Ch. VII]) and surrogate constraints [7]. These options will be further developed and evaluated in a future paper.

9. Computational Experience

In order to test the computational efficiency of the Accelerated Bound-and-Scan Algorithm, a FORTRAN code was written for the IBM 360/67 system by the first author. The two main parts of the code (1) obtain the Hermite normal form of A_1 and (2) execute the iterative portion of the algorithm in its skeleton version outlined in the preceding section. The Hermite normal form is obtained in a straightforward but relatively inefficient way, and a retrospective comparison with the computational experience (see [3]) of Bradley's algorithm [1] for doing this indicates that rewriting this part of the code could reduce the executive times to approximately 10% of their current values. The iterative portion of the algorithm was encoded with somewhat more care. The initial feasible solution for the algorithm has been obtained by using an available code for a basic heuristic procedure developed by the second author [13]. Part of the work required to set up the algorithm (primarily finding the vertices of the n-simplex S) is done by borrowing the initial part of the code for the Bound-and-Scan Algorithm [12]. When tabulating computational results, the total time required by the algorithm is broken into the following four components: (1) LP = time required by the simplex method to solve the continuous version of the problem, (2) Heur. = remaining time required by the heuristic procedure to find a good feasible solution to initiate the algorithm, (3) Setup = remaining time required to set up the algorithm (mostly finding the Hermite normal form of A_1), and (4) Iter. = time required to execute the iterative portion of the algorithm in its skeleton version.

The code has been used to solve a number of test problems taken from the literature. These include some of Haldi's "fixed-charge problems" (FC) and "IBM test problems" (IBM), as well as Woolsey's 5-point "combinatorial

problem" (COMB-5), all of which are reproduced by Trauth and Woolsey [15]. In addition, the code was applied to two of Petersen's problems [14] (PET) of the Lorie-Savage capital budgeting variety. The results are shown in Table I,* where the size of each problem is identified by m, the number of functional constraints, and n, the number of original variables (excluding slack and artificial variables). The times shown in parentheses are those that would have been obtained if Bradley's code for obtaining the Hermite normal form of A_1 in a more efficient way [1] had been used instead, according to Bradley's reported times [3] on the same problems.**

Table I also gives the available execution times on these problems for several other algorithms: (1) Hillier's Bound-and-Scan Algorithm [12] run on the IBM-360/67 system, (2) Gorry and Shapiro's Adaptive Group-Theoretic Algorithm [10] run on the IBM-360/65 system, (3) Gomory's Fractional Algorithm [9] with Haldi and Issacson's LIP 1 code [11]*** run on the IBM-7090 computer, as reported by Trauth and Woolsey [15], (4) Gomory's All-Integer Algorithm [8] with Woolsey's IPSC code*** [16] run on the CDC-3600 computer, as reported by Trauth and Woolsey [15], and (5) Geoffrion's Implicit Enumeration Algorithm [6] run on an IBM-7044 computer. (A blank indicates that no time has been reported for that problem.) Geoffrion's algorithm is for the special case of 0-1 variables, so that (except for the Petersen problems) a binary representation of the variables was used as described in

* An attempt also was made to apply the code to two problems not shown in Table I: (1) IBM-4, which violated the assumption that the optimal non-integer solution is unique, and (2) COMB-4, where the simplex method obtained an integer-valued optimal solution so the algorithm was not needed.

** Bradley's code actually was run on a somewhat slower computer (the IBM-7040-7094), but this has been ignored to compensate for the fact that Setup includes slightly more than constructing the Hermite normal form of A_1.

***These codes have been chosen since, according to Trauth and Woolsey's data, they appear to be the best available for the respective algorithms.

[6]. His times were given in minutes to two decimal places, so they have been converted to seconds here simply by multiplying by 60, except that his reported times of 0.01 minute are given just as < 1 second. It should be emphasized that the algorithms were run on different computers, so the execution times are not directly comparable.

TABLE I

EXECUTION TIMES FOR STANDARD TEST PROBLEMS

			Accelerated Bound-and-Scan Algorithm (IBM-360/67)					Other Algorithms				
								Hillier B-and-S	Gorry-Shapiro Ad.G.T.	Gomory Fract. LIP 1	Gomory All-Int. IPSC	Geoffrion Imp.En.
Problem	m	n	LP	Heur.	Setup	Iter.	TOTAL	360/67	360/65	7090	CDC-3600	7044
FC-1	4	5	0.03	0.04	0.23 (0.03)	0.02	0.32 (0.12)		0.41	1.83	0.74	
FC-2	4	5	0.03	0.00+	0.23 (0.03)	0.01	0.27 (0.07)			1.35	0.89	
FC-3	4	5	0.03	0.05	0.23 (0.02)	0.02	0.33 (0.12)		0.82	1.88	1.01	
FC-4	4	5	0.03	0.03	0.22 (0.02)	0.00+	0.28 (0.08)		0.47	1.48	0.63	
FC-5	6	5	0.09	0.05	0.26 (0.03)	41.37	41.77 (41.54)			9.01	79.90	
FC-6	6	5	0.10	0.06	0.26 (0.03)	21.65	22.07 (21.84)			7.57	43.48	
FC-7	4	5	0.03	0.03	0.23 (0.03)	41.48	41.77 (41.57)			7.83	> 100	1.8
FC-8	4	5	0.03	0.05	0.22 (0.03)	21.51	21.81 (21.62)			6.42	> 100	3.0
FC-9	6	6	0.06	0.03	0.31 (0.03)	0.03	0.43 (0.15)		5.78	3.23	5.48	
IBM-1	7	7	0.08	0.03	0.41 (0.04)	0.01	0.53 (0.16)		0.24	1.87	0.92	< 1
IBM-2	7	7	0.08	0.06	0.41 (0.04)	0.00+	0.55 (0.18)		0.27	3.02	1.04	1.2
IBM-3	3	4	0.02	0.01	0.15 (0.01)	0.03	0.21 (0.07)	0.55	0.16	2.87	0.48	< 1
IBM-5	15	15	0.60	0.18	2.17 (0.18)	6.33	9.28 (7.29)			66.48	62.82	114
IBM-9	50	15	10.03	0.53	--	--	>100	14.99		473.10	95.39	26.4
COMB-5	20	10	0.79	1.30*	1.31 (0.10)	0.10*	3.50* (2.29)			19.80	5.37	
PET-4	10	20	1.62	0.40	2.89	0.19	5.10	2.79				2.4
PET-5	10	28	0.61	0.51	5.33	3.82	10.27	8.77				14.4

* For COMB-5, the heuristic procedure failed to find a feasible solution and a known optimal solution was substituted, so Iter. only gives the time required to verify optimality.

In order to further evaluate how the execution time for the Accelerated Bound-and-Scan Algorithm might tend to vary with problem size for problems of similar structure, the code also was applied to 16 randomly generated problems. Four of these were "Type III" problems used previously by the second author to test his heuristic procedures [13] and Bound-and-Scan Algorithm [12] (approximately 1.5 seconds total were required for the simplex method, a heuristic procedure, and this algorithm on each of these problems). The other 12 are new "Type V" problems whose parameters are integers randomly generated from the intervals indicated in Table II.

TABLE II

DESCRIPTION OF THE RANDOMLY GENERATED TEST PROBLEMS

	Problem type	
	III	V
c_j	[0,99]	[0,5]
a_{ij}	[0,1]	[0,5]
b_i	1	10 for V-1 to V-9 20 for V-10 40 for V-11 80 for V-12

The computational results for these problems are given in Table III. Since the problems proved to be "too easy" for the iterative portion of the algorithm, the most interesting results are the setup times and their very stable dependence on problem size.

TABLE III

EXECUTION TIMES FOR RANDOMLY GENERATED TEST PROBLEMS

Problem	m	n	LP	Heur.	Setup	Iter.	TOTAL
III-2	15	15	0.55	0.15	2.14	0.01	2.85
III-4	15	15	0.60	0.13	2.12	0.01	2.86
III-5	15	15	0.57	0.10	2.10	0.01	2.78
III-8	15	15	0.61	0.16	2.15	0.01	2.93
V-1	5	5	0.05	0.03	0.24	0.00+	0.32
V-2	5	10	0.07	0.08	0.70	0.64	1.49
V-3	5	15	0.11	0.14	1.41	0.03	1.69
V-4	5	20	0.13	0.13	2.40	0.00+	2.66
V-5	5	25	0.23	0.29	4.03	0.05	4.60
V-6	5	40	0.23	0.70	11.25	0.07	11.25
V-7	10	15	0.31	0.11	1.83	0.01	2.26
V-8	20	15	1.47	0.33	2.62	0.01	4.42
V-9	40	15	6.91	0.22	5.28	0.03	12.44
V-10	10	15	0.31	0.20	1.80	0.03	2.34
V-11	10	15	0.29	0.20	1.80	0.03	2.32
V-12	10	15	0.30	0.19	1.79	0.03	2.31

10. Conclusions

A basic consideration motivating both the Bound-and-Scan and Accelerated Bound-and-Scan Algorithms is the availability of heuristic procedures such as those developed by the second author [13]. The heuristic procedures normally can find a very good (frequently optimal) feasible solution extremely quickly. Since the number of better feasible solutions should be relatively small, an

optimal solution can then be identified efficiently if these better feasible solutions can be directly enumerated in an efficient way. Both algorithms are carefully designed to attempt to efficiently enumerate a sequence of improving feasible solutions until no better ones remain. Computational experience obtained from unrefined computer codes of simple versions of these algorithms has been encouraging to the authors, suggesting that this basic approach may have considerable potential.

The Bound-and-Scan Algorithm constructs each new improved feasible solution by successively assigning appropriate integer values to the variables, beginning with the variables that are nonbasic in the optimal linear programming solution and concluding with the basic variables. The construction procedure for the nonbasic variables is extremely efficient, and it generates only those partial solutions which have improved completions (perhaps non-integer) that satisfy all of the binding constraints (5). On the other hand, the procedure for the basic variables is much less efficient, and it can generate "excess" partial solutions having no such improved completions. The Accelerated Bound-and-Scan Algorithm attempts to rectify this by using a transformation of variables that may increase the number of nonbasic variables, and that also allows applying a slight modification of the procedure for nonbasic variables to the basic variables. This tends to greatly accelerate the iterative portion of the algorithm *. The price that is paid for this acceleration is the extra setup time to perform this transformation by constructing the Hermite notmal form. This extra time is fairly well predictable. However, as indicated in Tables I and III, this

* An acceleration is not guaranteed, however, since the transformation of variables can increase the computational effort required to generate the same completions, e.g., by generating a value of B_{ii} for $i = n_1+1$ that is tremendously larger than those for $i > n_1+1$.

setup time is significant for problems of moderate size (even recognizing that the times given are an order of magnitude larger than necessary with the best available procedure and code). Furthermore, if the elements of A_1 are large, then the code for constructing its Hermite normal form may fail because of overflow. Therefore, the Accelerated Bound-and-Scan Algorithm should not automatically be preferred to the original Bound-and-Scan Algorithm

Available comparative computational experience is given in Table I. It appears that the current code for the skeleton version of the Accelerated Bound-and-Scan Algorithm is at least competitive with the other algorithms, and this would be even more true if the best available code were used for the setup (see the times in parentheses). However, it has often been observed that there probably will never be an integer programming algorithm that is superior to all others on all problems, and that certainly continues to be the case here. The apparently best available codes (according to Trauth and Woolsey's data [15]) for Gomory's Fractional and All-Integer Algorithms performed better on some of the difficult problems. The Gorry-Shapiro algorithm tended to be slower on the easy probelms but, unfortunately, a meaningful comparison is not possible since Gorry and Shapiro did not report any times on the standard difficult problems. Geoffrion's algorithm was an impressive performer on a slower computer. It continues to appear that a good 0-1 algorithm (and this is a very good one) should tend to be superior to a general algorithm on a 0-1 problem, or perhaps even on a general integer programming problem having very small upper bounds on the variables. A comparison with the Bradley-Wahi algorithm for the cone problem has been given in Section 7.

However, one extremely important current advantage of the Accelerated Bound-and-Scan Algorithm over the others is that its computation time can be quite accurately estimated in advance for any

given problem. The first author [4] presents methods for doing this, as well as establishing upper bounds on computational effort, in a companion paper Therefore, the user can learn in advance whether this algorithm will solve his problem in a reasonable length of time or another algorithm should be tried instead.

Another attractive feature of this algorithm is that it is primal in nature. It starts with a good feasible solution and then successively finds better ones. Therefore, if the algorithm must be terminated before completion, a near-optimal solution will still be provided.

These two features suggest another interesting possibility for the use of this algorithm. Suppose that the estimated execution time for a given problem is excessive, and it appears that the same may be true for other available algorithms as well (perhaps because the problem is large without special structure). If the optimal solution is only slightly better than the feasible solution provided by heuristic procedures, then it may not be worth a long computer run to try to improve upon this feasible solution. Therefore, the Accelerated Bound-and-Scan Algorithm can be used instead to find the optimal solution only if it is substantially better, and otherwise verify that it is not. This is done simply by setting some strictly positive lower bound $\underline{\rho}_0$ on the values ρ_0 that will be considered, so that the algorithm will seek only feasible solutions x such that $c^T x \leq c^T x^{(F)} + \underline{\rho}_0[c^T x^{(0)} - c^T x^{(F)}]$, where $x^{(F)}$ is the feasible solution provided by heuristic procedures and $x^{(0)}$ is the optimal solution to the linear program (1,2,3). If $\underline{\rho}_0$ is significantly greater than zero, this can reduce execution time dramatically.

There remain a number of important possibilities for improving and generalizing the Accelerated Bound-and-Scan Algorithm (or the Bound-and-Scan

Algorithm) that are being investigated by the authors. These include the additional options for the algorithm mentioned at the end of Section 8. Also, the ordering of the rows of A_1 can greatly affect computational effort by changing the relative magnitudes of the B_{ii} for $i > n_1$, and the implications of this are being studied. The algorithm's assumption that $x^{(0)}$ is unique is an unfortunate restriction, and some ways of eliminating this restriction are being explored, including those proposed for similar algorithms by the second author [12] and by Bradley and Wahi [3]. Finally, high priority will be placed on extending the algorithm to the mixed integer linear programming problem.

REFERENCES

1 BRADLEY, Gordon H., "Algorithms for Hermite and Smith Normal Matrices and Linear Diophantine Equations", Mathematics of Computation, Oct., 1971.

2. BRADLEY, Gordon H , "Equivalent Integer Programs and Canonical Problems", Management Science, Vol. 17, No. 5 (1971), pp 354-366.

3. BRADLEY, Gordon H and WAHI, Pran N., "An Algorithm for Integer Linear Programming: A Combined Algebraic and Enumeration Approach", Report No. 29, Department of Administrative Sciences, Yale University, December 16, 1970 (Revised February, 1971).

4 FAALAND, Bruce, H., "Estimates and Bounds on Computational Effort in the Accelerated Bound-and-Scan Algorithm", Technical Report #35 (submitted to Operations Research).

5. FAALAND, Bruce H , "Some Algebraic and Number Theoretic Methods in Integer Programming", Ph D. dissertation, Stanford University, 1971.

6. GEOFFRION, A.M , "An Improved Implicit Enumeration Approach for Integer Programming", Operations Research, Vol 17 (1969), pp 437-454.

7. GLOVER, Fred, "Surrogate Constraints", Operations Research, Vol. 16, No. 4 (1968), pp 741-749.

8. GOMORY, Ralph E., "All-Integer Integer Programming Algorithm", in J.F. Muth and G.L. Thompson (eds.), Industrial Scheduling, pp 193-206, Prentice-Hall, New York, 1963, first issued in 1960

9. GOMORY, Ralph E , "An Algorithm for Integer Solutions to Linear Programs", in R.L. Graves and P. Wolfe (eds.), Recent Advances in Mathematical Programming, pp. 269-302, McGraw-Hill, New York, 1963; first issued in 1960

10. GORRY, G. Anthony, and SHAPIRO, Jeremy F , "An Adaptive Group Theoretic Algorithm for Integer Programming Problems", Management Science, Vol. 17 (1971), pp 285-306.

11 HALDI, John, and ISAACSON, Leonard M., "A Computer Code for Integer Solutions to Linear Programs", Operations Research, Vol 13 (1965), pp 946-959.

12 HILLIER, Frederick S , "A Bound-and-Scan Algorithm for Pure Integer Linear Programming with General Variables", Operations Research, Vol. 17, No 4 (1969), pp 638-679.

13. HILLIER, Frederick S., "Efficient Heuristic Procedures for Integer Programming with an Interior", Operations Research, Vol. 17, No. 4, (1969), pp 600-637.

14. PETERSEN, Clifford C., "Computational Experience with Variants of the Balas Algorithm Applied to the Selection of R&D Projects", Management Science, Vol. 13 (1967), pp. 736-750.

15. TRAUTH, C.A., Jr., and WOOLSEY, R.E., "Integer Linear Programming: A Study in Computational Efficiency", Management Science, Vol. 15 (1969), pp. 481-493.

16. WOOLSEY, R.E., and TRAUTH, C.A., Jr., "A Machine Independent Integer Linear Program", Sandia Laboratories Research Report SC-RR-66-433, July, 1966.

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA • R&D
(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Department of Operations Research Stanford University Stanford, California	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE
The accelerated bound-and-scan algorithm for integer programming

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
Technical Report

5. AUTHOR(S) (Last name, first name, initial)
Bruce H. Faaland and Frederick S. Hillier

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
May 15, 1972	37	16

8a. CONTRACT OR GRANT NO. N00014-67-A-0112-0058 b. PROJECT NO. (NR-047-061) c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) Technical Report No. 34 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) Technical Report No. 9 - NSF GJ-31521

10. AVAILABILITY/LIMITATION NOTICES
This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Operations Research Program (Code 434) Office of Naval Research Washington, D.C. 20360

13. ABSTRACT

This paper presents a new implicit enumeration algorithm for solving the pure integer linear programming problem. The theory of equivalent integer programming problems is first used to reformulate the problem. A technique, originally used with particular success in the bound-and-scan algorithm to deal with only a subset of the variables, is extended to all of the variables in the restructured problem. In addition to the resulting basic enumeration scheme, the algorithm includes a scanning procedure and a method for identifying constraints which become redundant during the course of the algorithm. Computational experience on standard test problems is reported.

DD FORM 1 JAN 64 1473

UNCLASSIFIED
Security Classification